UNDERSTANDING MICROSERVICES

VENKATT GUHESAN

MAY 2019

OUR GOAL TODAY

UNDERSTANDING OUR HUMAN JOURNEY WILL HELP YOU IDENTIFY WHERE WE ARE GOING AND HOPEFULLY UNDERSTAND THE PLACE FOR MICROSERVICES IN THIS JOURNEY AND THE FUTURE AHEAD

- Me ;-)

A little about me!

MICROSERVICES – DEFINITION (CURRENT)

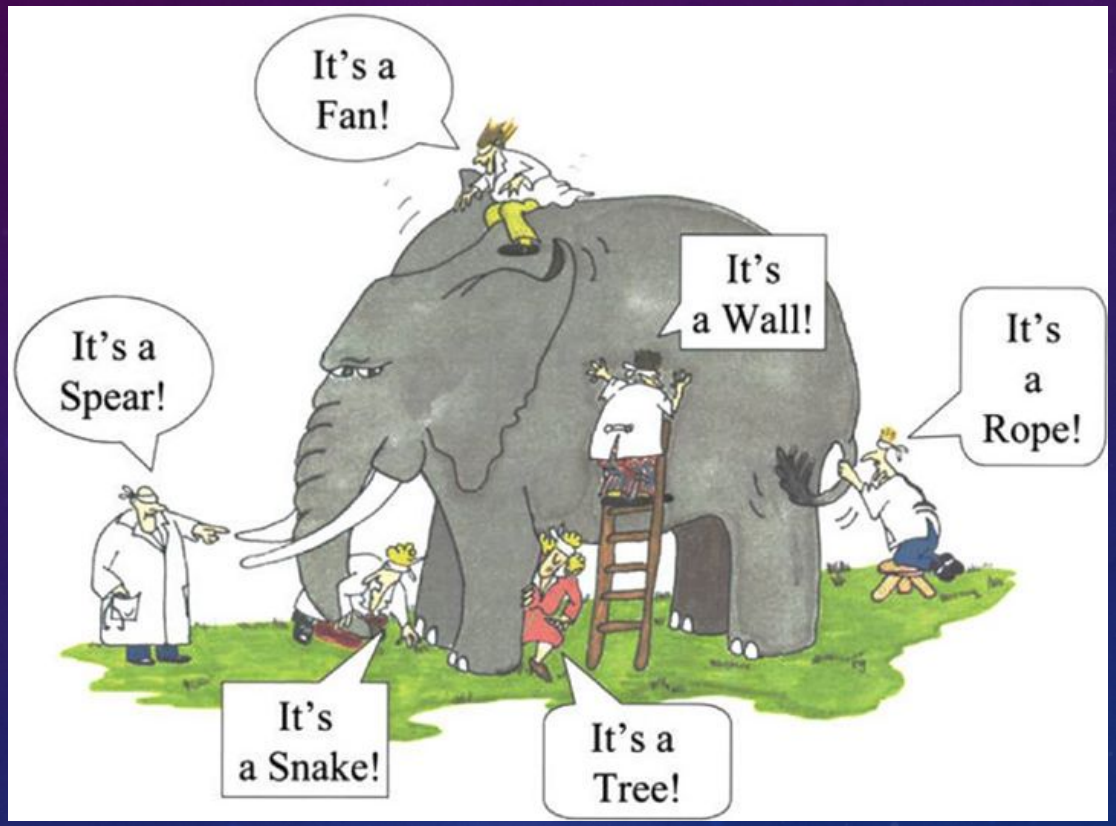
Microservices is a software development technique where the components (or parts of the application) are:

1. Loosely Coupled
2. Highly Maintainable
3. Independently Deployable

Three-Criteria applied:
Yesterday vs. Today vs. Tomorrow
== totally different results

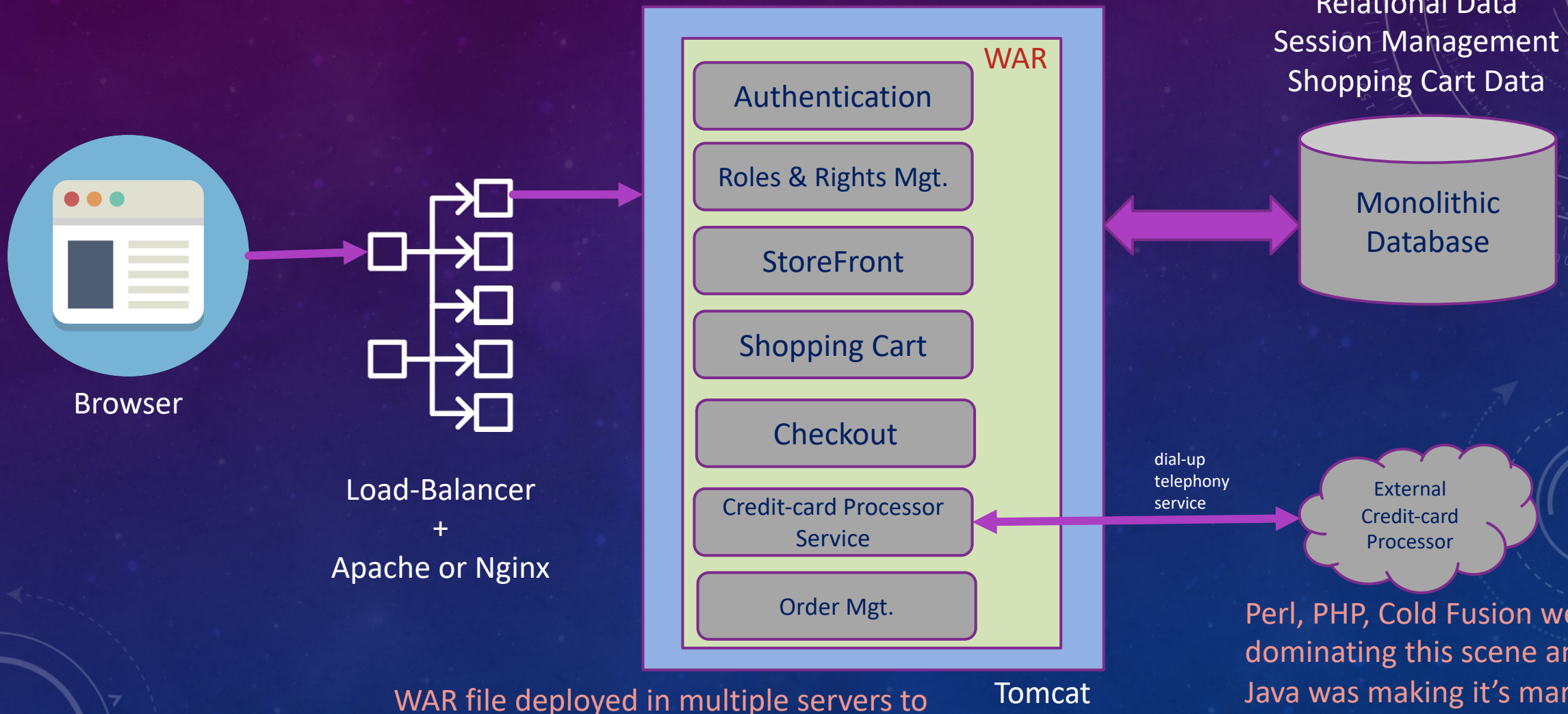
So any application you're developing that meets the three criteria can be technically called a "microservice".

MICROSERVICES – REALITY

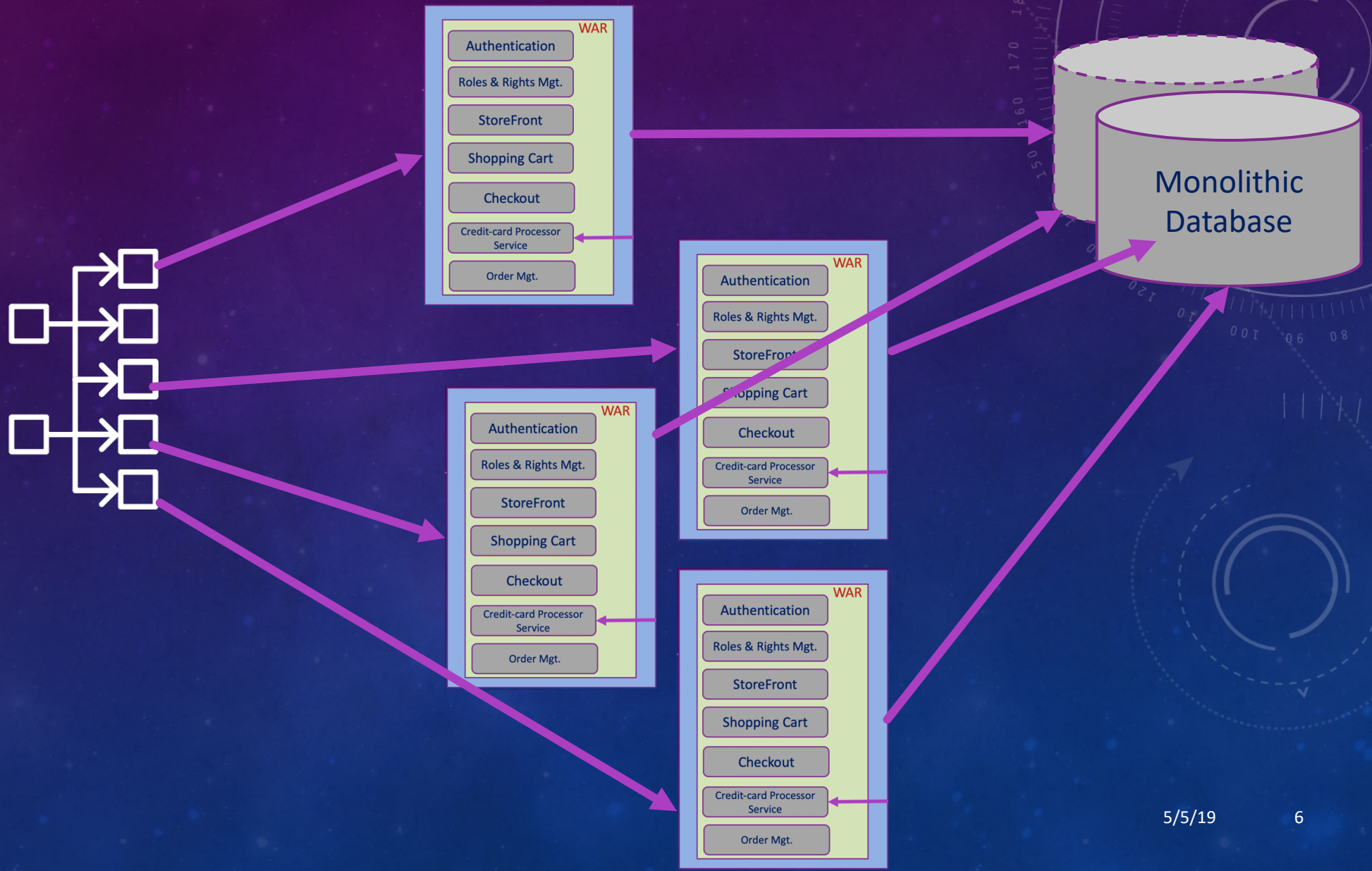


Let's take a look at how we got here in the next set of slides … The Journey

LEGACY MONOLITHIC APPLICATION STRUCTURE

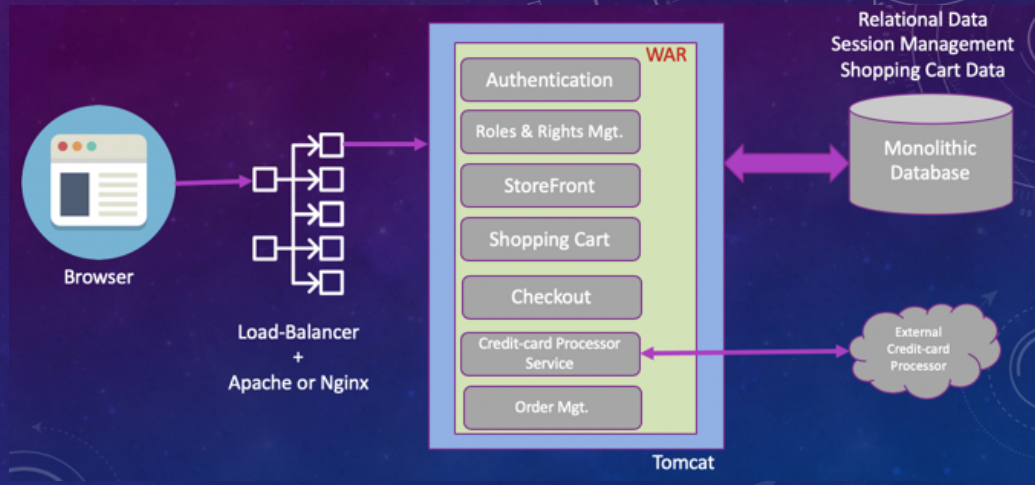


SCALING OUT A LEGACY MONOLITHIC APPLICATION



LEGACY MONOLITHIC APPLICATION DEFICIENCIES

- Threads were the bottleneck, this fueled …
- Servers with multi-CPUs and …
- Introduced the concept of multi-cores

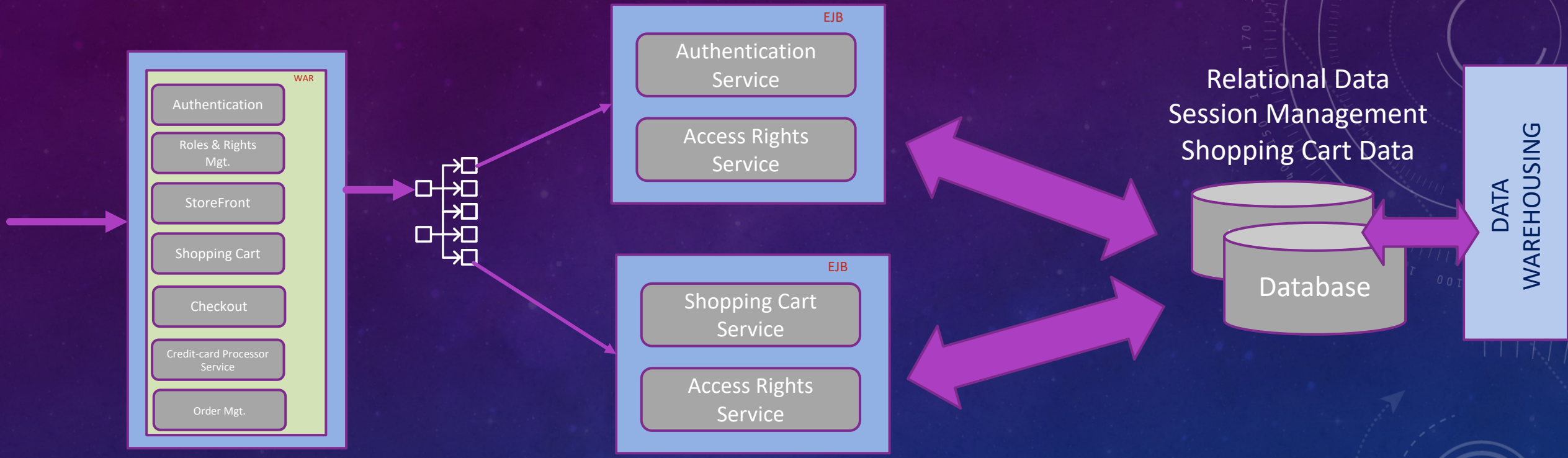


But fundamentally they were suffering from:

- ❑ Limited Concurrent-Processing
- ❑ Limited in capacity/clustering using Memcached and other similar disk-persistence technologies
- ❑ Easy prey to DoS attacks (which proliferated the Load-Balancer market space)

Then came Containers/J2EE/EJBs

ENTERPRISE (J2EE) APPLICATION STRUCTURE



Tomcat

- J2EE Container
-
-

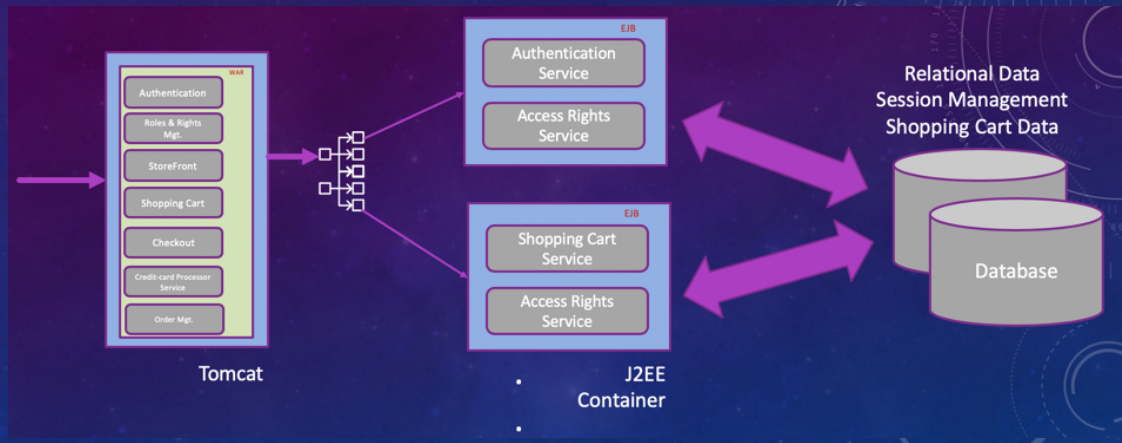
Perl, PHP, Cold Fusion are slowly being replaced!

Java is now used extensively on the server-side and led to the growth of the Enterprise

ENTERPRISE (J2EE) APPLICATION DEFICIENCIES

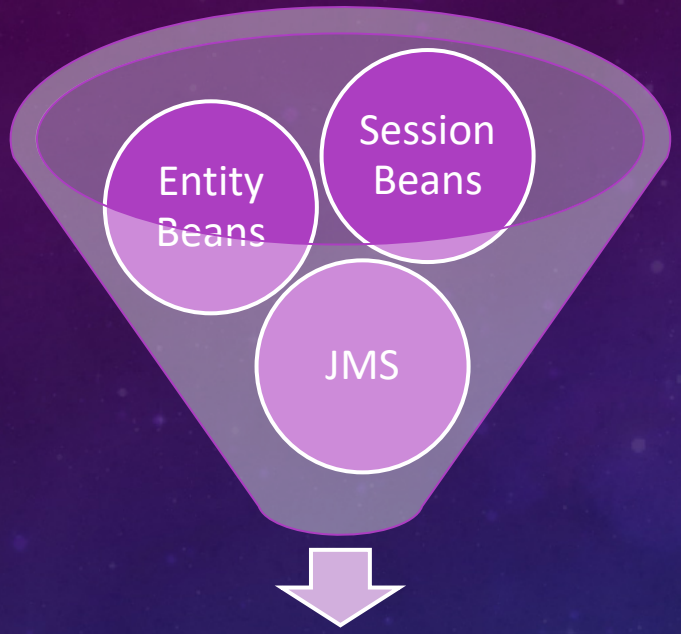
- Threads were still the bottleneck
- But then came multi-cores & multi-CPU's to push us forward (this meant more POSIX threads)
- Reaching 10K concurrent connections was still an issue.

But the underlying problem was shifted to multiple tiers there-by giving the illusion of scaling. But this new architecture gave rise to lookup services/RMI/JNDI



Then came Rod Johnson who introduced the Java world to Spring / DI / IoC… which brought us to the "age of the simple-container"

AGE OF THE SIMPLE-CONTAINER



Simple (Web/REST) Container

- ✓ Java was the ruler in this domain!
- ✓ Dependency Injection & Inversion of Control meets modern world
- ✓ Simplified Software Development & Deployment

But in this world how does one SCALE UP?

The World looks back in time

&

Erlang comes to the rescue with Actor-Message

("little engine that could" from the telecom era)

Welcome to the world of Queues

- RabbitMQ, ZeroMQ
- MuleSoft
- Camel etc.

➔ Enterprise Software Bus [ESB]

In between all of this the web hosting world was shifting to a CLOUD

BIRTH-OF-THE-CLOUD VS THE ENTERPRISE

Cloud

- The generation that was left behind in the web world with PHP, Cold Fusion, Perl got a new boost with the cloud. They began reinventing the old tools in a new paradigm.
- Newer easy to deploy, easy to spin off new instances and tool sets such as Python, Ruby began to populate the landscape.
- Ruby on Rails, Django for Python, CakePHP for PHP were showing off how easy it is to create applications
- To compete with this Java introduces Groovy (and Grails) … this revolution eventually takes us to POLYGOT software development! Scripting language support gets introduced into Java

Exponential growth of data

@ The Enterprise

- Data storage solutions increases to support the increased data growth
- Map-Reduce comes to the rescue for searching through the forest for a needle.
- Distributed File systems takes center-stage introducing Peta Bytes and Yeta Bytes into our everyday vocabulary
- Java solves the challenges with Hadoop, HDFS

DATA BECOMES THE KING

To support this new wave, there is a silent revolution going on in the back-drop within the storage world with iSCSI, de-duping, multipath, RAID-10, Fiber-Channel, Infiniband, dNFS, ZeroCopy, etc.

DATA MEETS THE CLOUD… EVENT-LOOP/KQUEUE

With better tools and services for easy deployment folks began experimenting with technologies that dominated the browser-side such as:

- V8 engine for JavaScript leveraging the event-loop
- Services like Heroku, Digital Ocean began to take over.
- World of Cloud Applications and scripting languages like Python, Ruby, Closure began to proliferate
- To complete with that Java secret weapon:
  - Spring Boot > Micronaut
  - GraalVM (more on this in the next slides)

Node JS and similar languages solved the 10K problem but then inherited limitations within the run-time engine and context switching!

- Everyone can own a piece of the pie and the BIG DREAM
- Spinning new instances for scaling up and down became the norm
- This laid the perfect breading ground for …

MICROSERVICES

Small loosely coupled, highly maintainable, independently deployable pieces of components that makes up a BIG application

THE COMMON THREAD

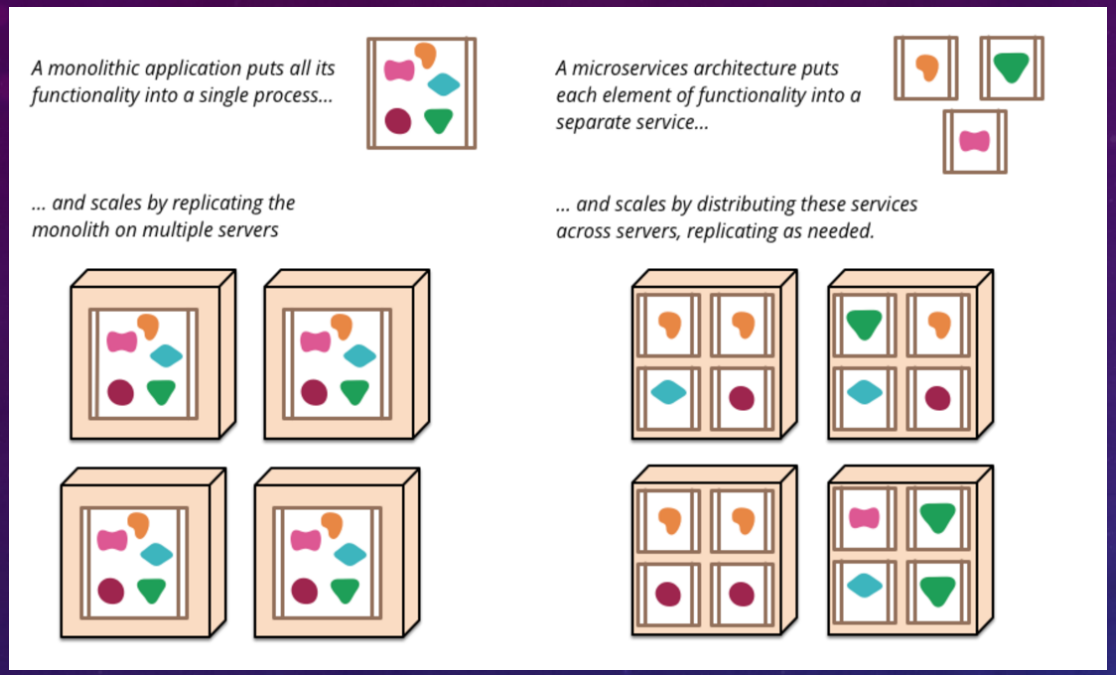
THROUGHOUT THIS JOURNEY THERE HAS ALWAYS BEEN A NEED FOR A COMMON SET OF CORE PRINCIPLES AND SERVICES TO MAKE THIS TECHNOLOGY EVOLUTION POSSIBLE

- ❑ Configuration Management
- ❑ Service Discovery & Lookup
- ❑ Load Balancing
- ❑ API Gateway (service gateway)
- ❑ Security
- ❑ Logging > (centralized logging)
- ❑ Application Tracing (distributed tracing)
- ❑ Debugging
- ❑ Fault-Tolerance (Resilience)
- ❑ Packaging, Deployment, Scheduling
- ❑ Job Management

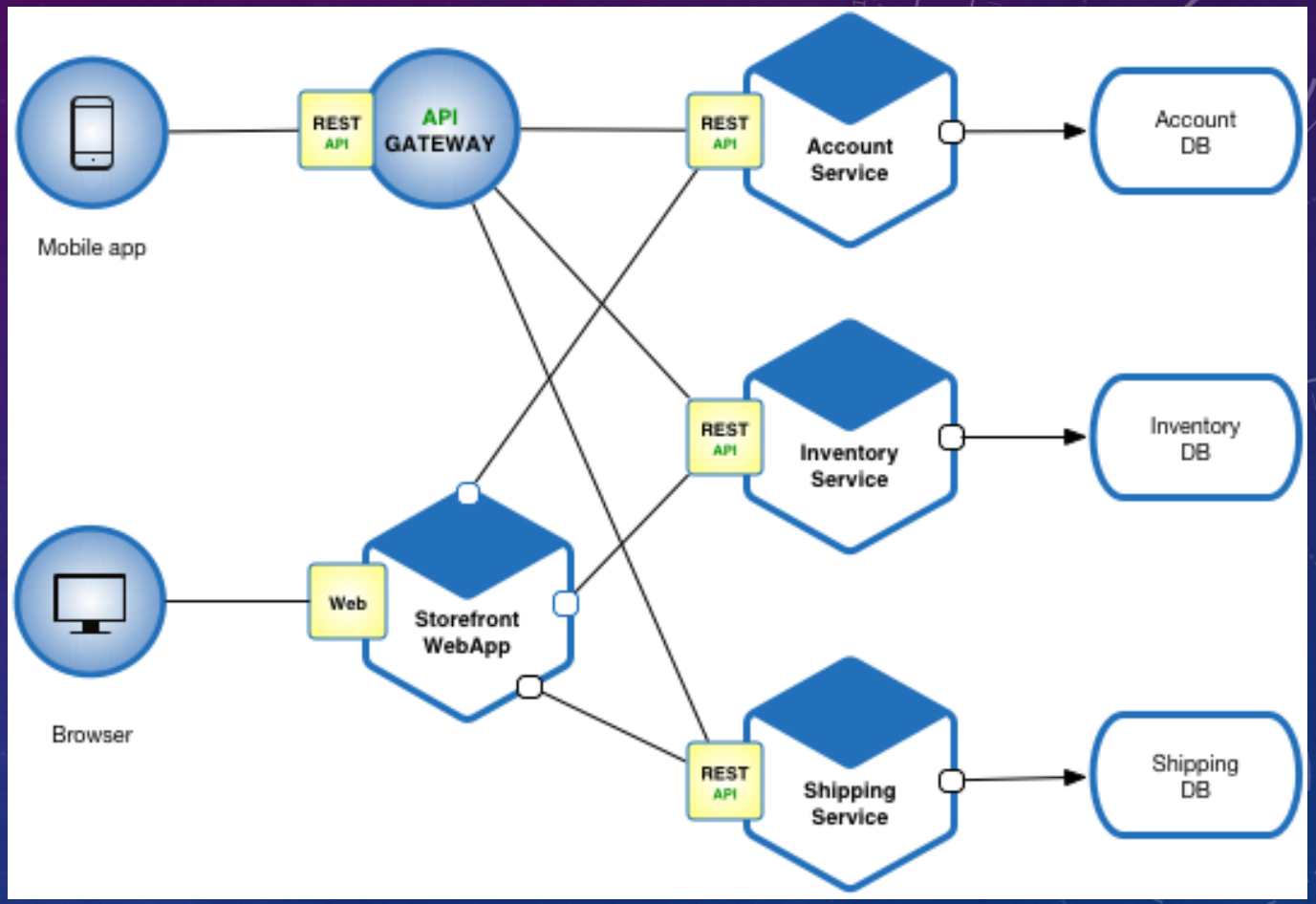
The very same concerns that existed since the microbial/mainframe days still exist today in microservices but how we solve them in the world of microservices is evolving!

How we leverage this trend and use it to our needs defines our prosperity!

MICROSERVICES AT 30,000 FEET



https://martinfowler.com/articles/microservices.html



https://microservices.io/

MICROSERVICES – THE CURRENT TREND

- Configuration Management ..... Spring Config
- Service Discovery & Lookup ..... Netflix Eureka/etcd/Consul/ZooKeeper
- Load Balancing ... Nginx / F5 BIG IP/ Netflix Ribbon / Riot Games / AWS Route 53 / RedBird
- API Gateway (service gateway) ..... Application Specific (REST API) – Tyk, Kong, Gravitee
- Security ..... Application Specific Dominance (Spring Cloud Zuul)
- Logging > (centralized logging) ..... ELK Stack / Splunk
- Application Tracing (distributed tracing) ..... Spring Spectator & Atlas / Grafana | Zipkin | Jaeger
- Debugging ..... GrayLog, ELK, Splunk
- Fault-Tolerance (Resilience) ..... Spring Hystrix, Turbine, Ribbon , Kong, Resilience4j
- Packaging ..... Native to language, Java > Maven, Node > npm, etc.
- Deployment ..... container-specific, cloud-specific
- Scheduling ..... Kafka, Walmart Labs BigBen, Celery Job queue, Risque for Ruby
- Job Management ..... SLURM, Torque, PBS Pro, Open Lava
- Authentication ..... Kong/OAUTH/JWT

EVOLVING TOOL CHAINS EXAMPLE

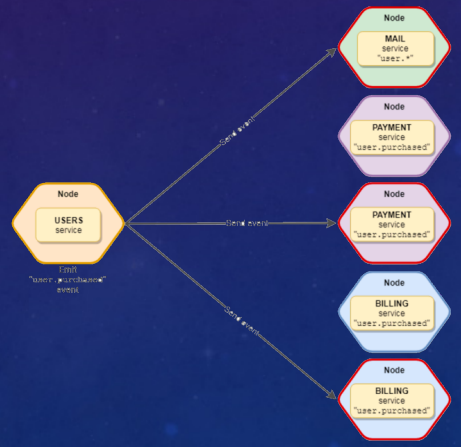
- Service Discovery & Registry: Netflix Eureka → ??? (Netflix has discontinued development – hence the proliferation of new services)
- Deployment: Netflix Asgard (deprecated) → Spinnaker (red/black | blue/green deployments)

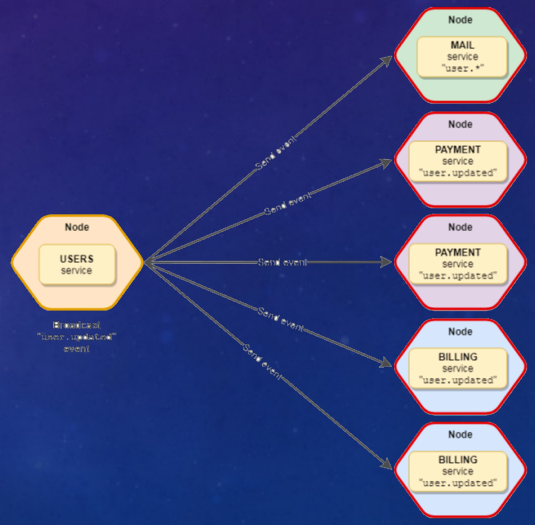
With so many tools to learn – which basket should I put my eggs?

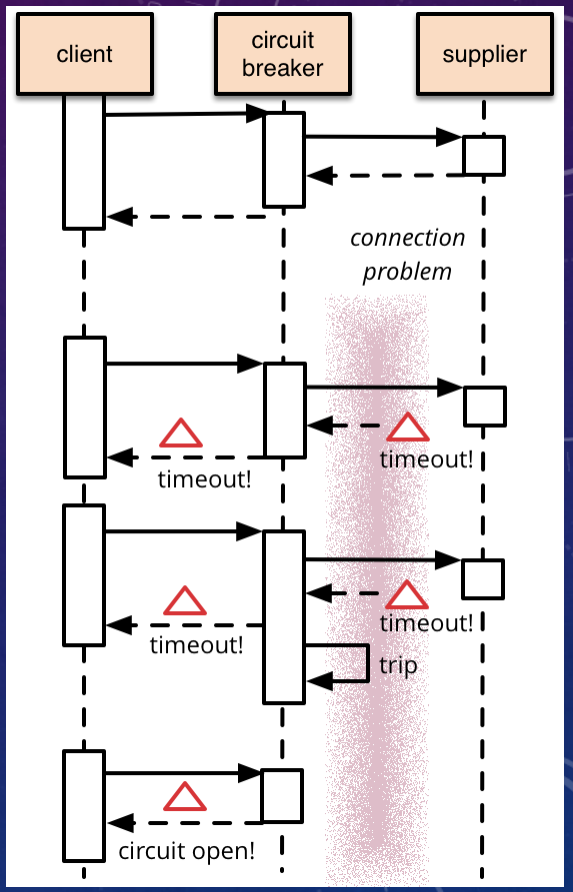
- ✓ Focus on learning the patterns that serve the Microservices architecture rather than a specific framework
- ✓ Let the technology-leads at large-organizations cherry-pick the grandiose pieces
- ✓ Since we are a Java-shop, learn Spring-Boot, Micronaut and GraalVM
- ✓ By the time the dust settles on the Java front wrt microservices, it will be a totally different landscape from where we are today. So understanding them at a high-level is key (not the details – we are focusing on the broad strokes and not the finer details)
- ✓ Focus on a much simpler model (ZeroConfig) where a lot of the ground-work has been simplified for you such as Node.js/JavaScript based Microservices frameworks like Molecular or Cote (this will help you get a better understanding of the parts and how they are evolving)

COMMON MICROSERVICES PATTERNS

- Circuit-Breaker pattern
- Rate-Limiter pattern
- Bulkhead (limit amount of parallel executions)
- Retry
- Cache
- Time Limit
- Health
- {Balanced, Broadcast} Events



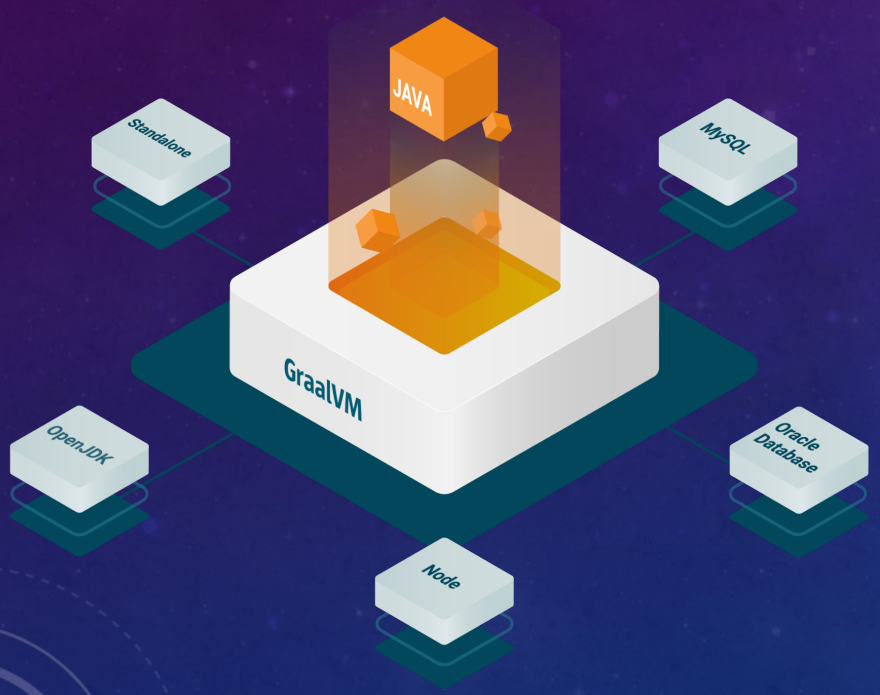


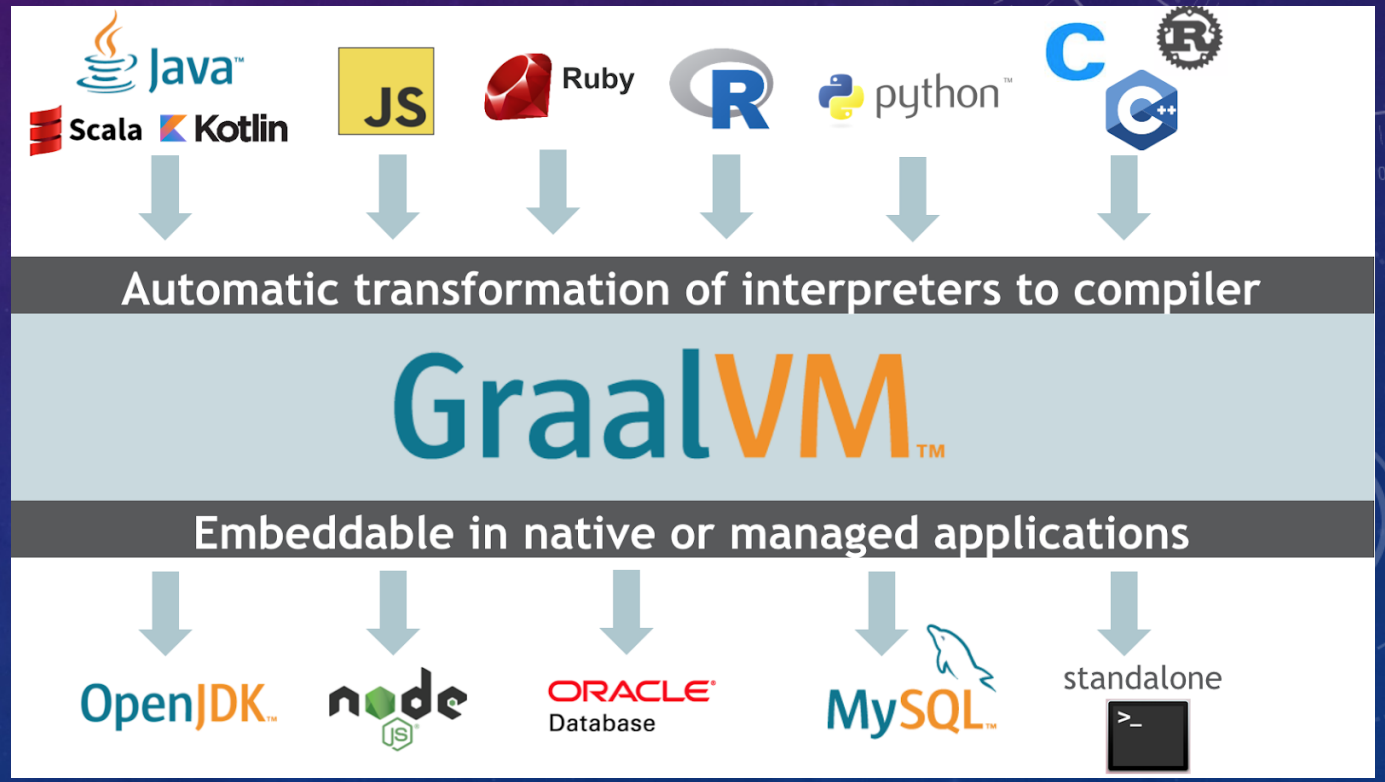


JAVA GRAALVM – FUTURE DEFINED

Run Programs Faster Anywhere

GraalVM is a universal virtual machine for running applications written in JavaScript, Python, Ruby, R, JVM-based languages like Java, Scala, Kotlin, Clojure, and LLVM-based languages such as C and C++.
GraalVM removes the isolation between programming languages and enables interoperability in a shared runtime. It can run either standalone or in the context of OpenJDK, Node.js, Oracle Database, or MySQL.





JAVASCRIPT – NODE.JS – COTE.JS

JavaScript / Node.js Based

- http://cote.js.org/
- https://moleculer.services/

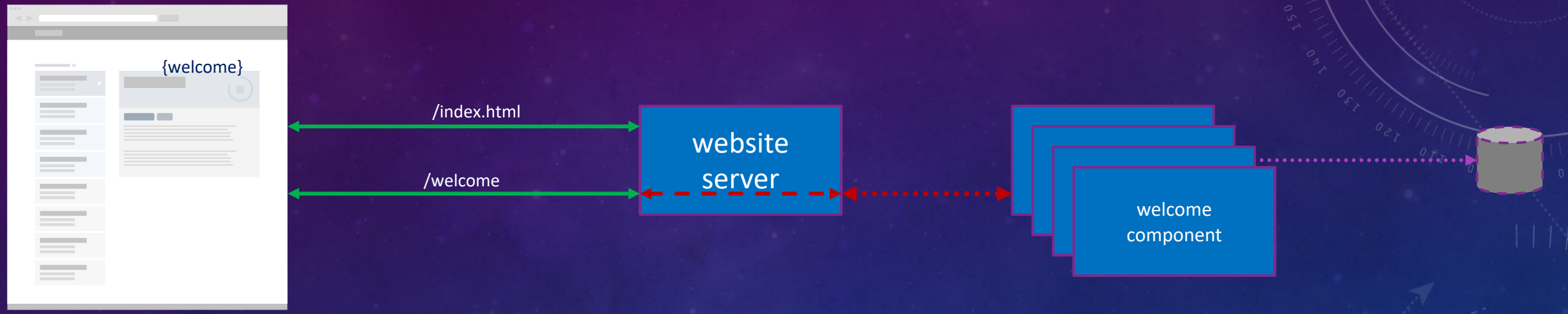
Java-Based

- https://micronaut.io/

DEMO TIME

- Moleculer - http://cote.js.org
  Javascript engineered for microservices

Demo



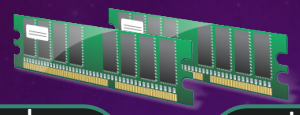
git clone https://vguhesan@bitbucket.org/vguhesan/understanding-microservices-demo.git

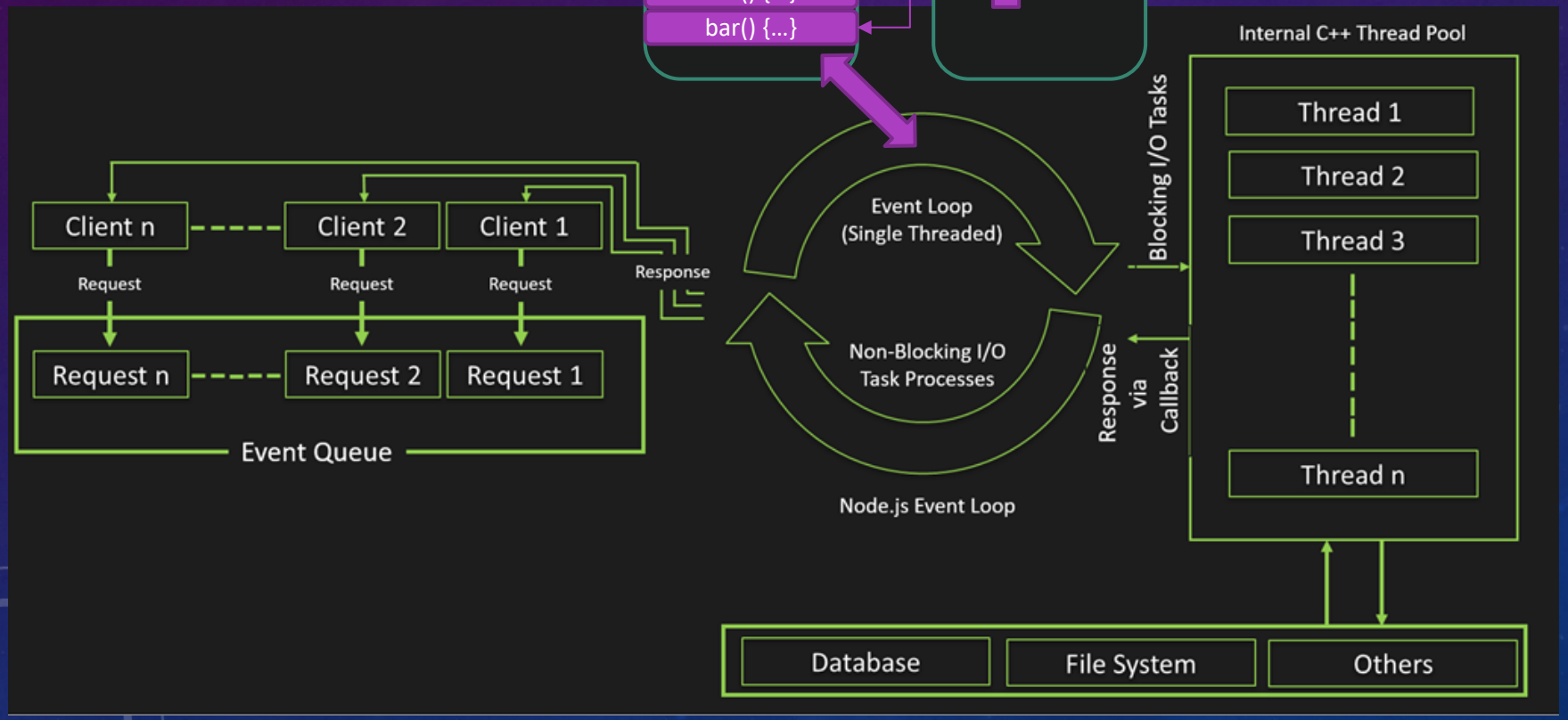
NODE.JS MISUNDERSTOOD

How does node.js scale using the event loop?

Shared resources include:

- ❑ CPU
- ❑ Network
- ❑ Network interfaces
- ❑ Memory
- ❑ Disk






```
1  function foo(b) {
2    var a = 10;
3    return a + b + 11;
4  }
5
6  function bar(x) {
7    var y = 3;
8    return foo(x * y);
9  }
10
11 console.log(bar(7)); //returns 42
```


IT’S TIME TO LEAVE
YOUR OWN
FOOTPRINTS WITH
MICROSERVICES